Development Process Documentation

### **# Prioritize Features:**

#### **1. Developer Experience:**

* Aiken is designed to be easy to learn and use, with clear error messages and a focus on developer productivity. This is inspired by modern languages like Gleam, Rust, and Elm.

#### **2. Robustness:**

* Since Aiken is specifically for smart contracts on the Cardano blockchain, security and reliability are paramount. The language is built to minimize errors and enable thorough review, audit, and static analysis of code.

#### **3. Focus on Cardano:**

* Unlike general-purpose languages, Aiken is built with Cardano in mind. It provides a high-quality toolkit specifically for developing reliable smart contracts on the Cardano blockchain.

#### **4. Simplicity and Manageability:**

* + Considering the critical nature of smart contracts, Aiken keeps the language itself small and focused. This allows for easier understanding and reduces the risk of introducing vulnerabilities.

### **# Feature Design:**

Aiken-lang's feature design centers around two key areas: **developer experience** and **robustness for smart contracts**. Here's a breakdown of some potential design choices based on this focus:

#### **Developer Experience:**

* **Type Safety:** Aiken could enforce type annotations to prevent runtime errors and make code easier to understand and maintain.
* **Functional Programming Paradigm:** This paradigm can simplify reasoning about program behavior and potentially reduce bugs.
* **Clear and Concise Syntax:** Aiken might borrow from languages like Rust or Elm to have a clean and readable syntax that reduces boilerplate code.
* **Excellent Error Messages:** Informative error messages can help developers pinpoint issues quickly and efficiently.
* **Rich Standard Library:** A well-stocked library with pre-built functions and data structures can save developers time and effort.

#### **Robustness for Smart Contracts:**

* **Immutability:** Encouraging immutable data structures can prevent accidental modification of state and potential security vulnerabilities.
* **Formal Verification:** Aiken might integrate with tools for formal verification, allowing developers to mathematically prove the correctness of their code.
* **Resource Management:** The language could have built-in mechanisms for managing resources like memory and computation to prevent bugs and denial-of-service attacks.
* **Access Control:** Features like fine-grained access control can help ensure only authorized users can interact with the smart contract.

#### **Additional Considerations:**

* **Interoperability:** Aiken might allow integration with existing Plutus code or other languages for broader functionality.
* **Testing Frameworks:** Support for robust testing frameworks can be crucial for catching bugs before deployment.

### **# Coding Standards:**

While Aiken-lang has a focus on developer experience, there isn't a formal document outlining a strict coding standard like some other languages. However, there are resources that can guide you in writing clean and secure Aiken code:

#### **Standard Library (stdlib):** The Aiken standard library (<https://github.com/aiken-lang/stdlib>) serves as a good reference. It showcases well-written and well-tested Aiken code, providing examples of idiomatic usage.

#### **Community Resources:** The Aiken project likely has a community around it. Searching for resources like community forums, discussions, or blog posts related to "Aiken coding practices" or "Aiken best practices" might reveal conventions or preferred coding styles within the community.

#### **General Smart Contract Security:** Since Aiken targets smart contracts, security best practices for smart contract development are generally applicable. Resources on secure smart contract coding in general can provide valuable guidance.

### 

### **# Development:**

Implement the new features according to their specifications.

Comprehensive testing documentation that includes test plans, test cases, and results for unit tests, integration tests, and performance tests of the developed features.

#### **Develop Test Plans:** Create detailed test plans for each new feature, specifying the testing objectives, scope, approach, resources, and schedule.

* Test Plan:

1. **Developed Features & Functions:** List of all functions available on Aiken

[Aiken-Plooty Milestone 2](https://docs.google.com/document/d/1_3Scz79BAnM_6V-r1xs_-mGZEbpWpMuu2y57OvdELE0/)

1. Developing New Features & Test Result: List of new functions we want to develop

[Aiken-Lang Functions Testing](https://docs.google.com/document/d/1-s1DhdvxLwcTl15cRFAOO_wdS1vTOz6z6QwG00vO0aY)

#### **Write Test Cases:**

Develop comprehensive test cases that cover all functional and non-functional aspects of the new features, including edge cases and failure scenarios.

#### Developed Functions test results & reports:

#### Development New Features results & reports:

### **# Execute Testing:** Conduct the testing as per the test plans, including:

#### **1. Unit Testing:** Aiken-lang boasts a built-in unit testing framework, making it convenient to write and execute tests directly within the language [2]. Here's a breakdown of unit testing in Aiken:

###### Concepts:

* + **Test Function:** A unit test is defined as a function with no arguments that returns a boolean value. The test passes if the function returns True, and fails otherwise.
  + **Placement:** You can write unit tests anywhere within an Aiken module, alongside your regular code. This allows for close integration and testing of specific functionalities.
  + **Execution:** The aiken check command is used to parse, collect, run all the tests in your code, and generate a report detailing the results.

###### Benefits:

* + **Same Virtual Machine:** Aiken unit tests leverage the same virtual machine used for on-chain smart contract execution. This ensures your tests run in the same environment as your deployed contracts, promoting code reliability [1].
  + **Reasoning:** Since tests are written in Aiken, you can leverage the language's features for reasoning about test behavior and expected outcomes. This simplifies the testing process [1].

#### **2. Integration Testing:**

To ensure that new features integrate seamlessly with existing features and the broader Cardano ecosystem.

#### **3. Performance Testing:**

To assess the performance improvements brought by the new features under various conditions.

#### **4. Testing Documentation:**

Compile the results from all testing phases into comprehensive documentation. This should include the context of the tests, how they were conducted, the results obtained, and any actions taken in response to the findings.

#### **5. Feature Documentation:**

Document the developed features, providing clear and concise information on their purpose, usage, and integration into the Aiken language. Include examples and best practices to assist developers in adopting these new features.

#### **6. Peer Review:**

**Aiken-lang** is a promising smart contract language designed for the Cardano blockchain. Here's a peer review based on available information:

**Strengths:**

* **Pure Functional:** Aiken promotes a functional programming paradigm, leading to potentially more predictable and easier to reason about code compared to imperative languages.
* **Small and Easy to Learn:** Aiken's focus on simplicity makes it approachable for developers new to smart contract development.
* **Strong Static Typing:** Static typing enforces type safety, reducing runtime errors and improving code reliability.
* **Modern Development Environment:** Aiken provides a zero-configuration development experience with features like auto-formatting and editor integration.
* **Baked-in Unit Testing:** Integrated unit testing facilitates writing robust smart contracts.

**Areas for Improvement:**

* **Limited Ecosystem:** As a relatively new language, Aiken might have a smaller developer community and fewer available libraries compared to established languages.
* **Cardano-Specific:** Aiken's focus on Cardano might limit its portability to other blockchains.
* **Security Considerations:** While Aiken promotes some security features, a thorough security audit is crucial for any smart contract language.

**Additional Considerations:**

* **Random Number Generation:** Cardano's lack of native Chainlink VRF support requires alternative approaches within Aiken for secure on-chain randomness.
* **Inheritance:** Cardano doesn't support inheritance for smart contracts. Aiken's documentation should clarify this limitation.

**Overall, Aiken-lang is a well-designed language with a focus on simplicity and security for Cardano smart contract development. As the language matures and the ecosystem grows, it has the potential to become a popular choice for blockchain developers.**

**For a more comprehensive review, consider these aspects:**

* **Performance:** Evaluate the efficiency of compiled Aiken code compared to other languages on Cardano.
* **Tooling:** Assess the maturity and capabilities of Aiken's development tools.
* **Community:** Gauge the size and activity of the Aiken developer community.
* **Real-world Usage:** Explore existing projects built with Aiken to understand its practical applications.

##### **References:**

1. <https://github.com/aiken-lang/aiken/discussions>
2. https://www.reddit.com/r/cardano/comments/11d9cuj/aiken\_simplifies\_writing\_smart\_contracts\_for/?rdt=62062

**On-Going These Sections**

# 

# Acceptance criteria

#### 1. Aiken language features (focus on marketplace smart contracts) adhering to the project's technical specifications and standards.:

Acceptance Criteria for Aiken Language Features in Marketplace Smart Contracts

Here are some acceptance criteria for Aiken language features focusing on marketplace smart contracts, adhering to project technical specifications and standards:

**General Functionality:**

* **Asset Management:** The language should support the creation, listing, and management of different asset types (e.g., tokens, NFTs) within the marketplace.
* **Escrow Functionality:** Aiken should enable secure escrow mechanisms to hold assets during transactions until specific conditions are met.
* **Offer Management:** The language should allow for creating, modifying, and canceling offers within the marketplace.
* **Dispute Resolution:** Features should support a dispute resolution process for potential conflicts between buyers and sellers.
* **Fee Management:** The language should enable defining and collecting fees associated with various marketplace activities (e.g., listing fees, transaction fees).
* **Authorization and Access Control:** Aiken should provide mechanisms to restrict access to specific marketplace functionalities based on user roles (e.g., admin, buyer, seller).

**Security:**

* **Reentrancy Protection:** The language should prevent reentrancy attacks that could exploit vulnerabilities in transaction logic.
* **Integer Overflow/Underflow Protection:** Aiken features should safeguard against integer overflow/underflow errors that could lead to unintended behavior.
* **Secure Randomness:** The language should integrate with Cardano's VRF functionality or offer alternative secure methods for on-chain randomness generation.

**Interoperability:**

* **Fungible Token Integration:** Aiken should seamlessly integrate with Cardano's native tokens for use as payment methods within the marketplace.
* **NFT Integration:** The language should support the creation and management of NFTs within the marketplace.
* **Oracle Integration:** If the marketplace relies on external data feeds (oracles), Aiken should allow for secure integration with oracle networks.

**Aiken-Specific Considerations:**

* **Code Readability:** Features should promote clear and concise code, leveraging Aiken's focus on functional programming for better maintainability.
* **Testability:** Language features should facilitate the writing of comprehensive unit tests for marketplace smart contracts, ensuring robust functionality.
* **Performance Optimization:** Aiken features should not introduce undue overhead, aiming for efficient code execution on the Cardano blockchain.

**Project-Specific Considerations:**

* **Compliance with Project Standards:** The chosen Aiken features should comply with any pre-defined coding standards or best practices established for the specific project.
* **Integration with Existing Tools:** Features should integrate seamlessly with any existing development tools or libraries used within the project.

**Additional Considerations:**

* **Scalability:** If the marketplace is expected to handle high transaction volume, the chosen Aiken features should be evaluated for scalability on the Cardano blockchain.
* **Future-Proofing:** Consider how well the chosen Aiken features can accommodate potential future developments or integrations within the marketplace.

By implementing these acceptance criteria, you can ensure that the chosen Aiken language features effectively support the development of secure, functional, and maintainable marketplace smart contracts that align with your project's technical specifications and standards.

#### 2. The developed features must go through testing phases, demonstrating reliability, efficiency, and compatibility with the Cardano ecosystem. For any features that do not pass testing, we should also mention why it did not:

You're absolutely right. Testing is crucial for ensuring the reliability, efficiency, and compatibility of Aiken language features, especially for marketplace smart contracts on Cardano. Here's how we can incorporate testing phases:

**Testing Phases:**

1. **Unit Testing:**
   * **Focus:** This initial phase focuses on testing individual Aiken functions within the marketplace smart contract. Mocks can be used to simulate interactions with other parts of the system.
   * **Acceptance Criteria:**
     + All core functionalities of the feature (e.g., asset creation, offer management, escrow) must work as intended in isolation.
     + Edge cases and potential error conditions (e.g., invalid inputs, insufficient funds) should be tested and handled gracefully.
   * **Reasons for Failure:** Features failing unit testing could indicate issues with the core logic of the Aiken code. This might require revising the feature's implementation or identifying limitations of Aiken itself for certain functionalities.
2. **Integration Testing:**
   * **Focus:** This phase tests how different Aiken features within the marketplace smart contract interact with each other. External dependencies like Cardano native tokens or oracles (if used) can be mocked or simulated.
   * **Acceptance Criteria:**
     + Features should work together seamlessly, ensuring a smooth user experience for marketplace operations (e.g., buying an NFT using a token).
     + Data flow and state transitions between features should be consistent and predictable.
   * **Reasons for Failure:** Integration testing failures might reveal issues with how features communicate or handle data. This could necessitate modifications to the interaction logic between features or potentially limitations in Aiken's capabilities for specific data exchange needs.
3. **System Testing:**
   * **Focus:** This final phase involves deploying the entire marketplace smart contract, written in Aiken, on a Cardano testnet environment. Here, the focus is on testing its real-world functionality and interaction with the Cardano ecosystem.
   * **Acceptance Criteria:**
     + The marketplace smart contract should function as intended with real Cardano transactions (e.g., listing assets, making offers, completing purchases).
     + Interactions with Cardano native tokens and (if applicable) oracles should be successful.
   * **Reasons for Failure:** System testing failures could indicate issues with compatibility between Aiken features and the Cardano environment. This might require exploring alternative approaches within Aiken or potentially limitations in the current Cardano testnet functionalities.

**Additional Considerations:**

* **Security Audits:** In addition to the testing phases, conducting independent security audits is highly recommended to identify and address potential vulnerabilities in the marketplace smart contract written in Aiken.
* **Test Coverage:** Aim for comprehensive test coverage throughout all phases to minimize the risk of unforeseen issues in production.
* **Documentation:** Thoroughly documenting test cases, expected results, and any identified failures is crucial for future reference and maintenance.

By implementing these testing phases, you can ensure that the chosen Aiken language features are reliable, efficient, and compatible with the Cardano ecosystem, leading to a robust and secure marketplace smart contract.

#### 3. The testing documentation must be complete and detailed, clearly describing the testing process, the tests conducted, the results obtained, and any issues identified and resolved.:

You're absolutely right. Testing is crucial for ensuring the reliability, efficiency, and compatibility of Aiken language features, especially for marketplace smart contracts on Cardano. Here's how we can incorporate testing phases:

**Testing Phases:**

1. **Unit Testing:**
   * **Focus:** This initial phase focuses on testing individual Aiken functions within the marketplace smart contract. Mocks can be used to simulate interactions with other parts of the system.
   * **Acceptance Criteria:**
     + All core functionalities of the feature (e.g., asset creation, offer management, escrow) must work as intended in isolation.
     + Edge cases and potential error conditions (e.g., invalid inputs, insufficient funds) should be tested and handled gracefully.
   * **Reasons for Failure:** Features failing unit testing could indicate issues with the core logic of the Aiken code. This might require revising the feature's implementation or identifying limitations of Aiken itself for certain functionalities.
2. **Integration Testing:**
   * **Focus:** This phase tests how different Aiken features within the marketplace smart contract interact with each other. External dependencies like Cardano native tokens or oracles (if used) can be mocked or simulated.
   * **Acceptance Criteria:**
     + Features should work together seamlessly, ensuring a smooth user experience for marketplace operations (e.g., buying an NFT using a token).
     + Data flow and state transitions between features should be consistent and predictable.
   * **Reasons for Failure:** Integration testing failures might reveal issues with how features communicate or handle data. This could necessitate modifications to the interaction logic between features or potentially limitations in Aiken's capabilities for specific data exchange needs.
3. **System Testing:**
   * **Focus:** This final phase involves deploying the entire marketplace smart contract, written in Aiken, on a Cardano testnet environment. Here, the focus is on testing its real-world functionality and interaction with the Cardano ecosystem.
   * **Acceptance Criteria:**
     + The marketplace smart contract should function as intended with real Cardano transactions (e.g., listing assets, making offers, completing purchases).
     + Interactions with Cardano native tokens and (if applicable) oracles should be successful.
   * **Reasons for Failure:** System testing failures could indicate issues with compatibility between Aiken features and the Cardano environment. This might require exploring alternative approaches within Aiken or potentially limitations in the current Cardano testnet functionalities.

**Additional Considerations:**

* **Security Audits:** In addition to the testing phases, conducting independent security audits is highly recommended to identify and address potential vulnerabilities in the marketplace smart contract written in Aiken.
* **Test Coverage:** Aim for comprehensive test coverage throughout all phases to minimize the risk of unforeseen issues in production.
* **Documentation:** Thoroughly documenting test cases, expected results, and any identified failures is crucial for future reference and maintenance.

By implementing these testing phases, you can ensure that the chosen Aiken language features are reliable, efficient, and compatible with the Cardano ecosystem, leading to a robust and secure marketplace smart contract.

# Evidence of milestone completion

**Code Repository Links:**

1. <https://github.com/Plutus-art/Catalyst-Research>
2. https://github.com/Plutus-art/plutus-art-aiken-smartContract

**Testing Reports:**

1. https://docs.google.com/document/d/1-s1DhdvxLwcTl15cRFAOO\_wdS1vTOz6z6QwG00vO0aY/edit

**Development Documentation:**

Here will be the project catalyst link